**IN THE CLAIMS:**

1. (currently amended) A method for reducing contention of a highly contended software lock protecting data items of a data set, all of the data items being stored in a system memory of a multi-processor computer system, said method comprising the steps of:

defining partitions within the data set;

creating N partition locks, one for each partition, where $N \geq 2$;

identifying one code path from one or more code paths of a software program that access one or more of the data items;

determining which data items of the data set are ~~touched~~ accessible by the identified code path;

partitioning at least a portion ~~some~~ of the data items that are ~~touched~~ accessible by the identified code path into the partitions; and

optimizing ~~the~~ locking requirements of the identified code path so the locks being acquired and released in the identified code path are those associated with the data items being ~~touched~~ accessible by the identified code path.

2. (original) The method according to claim 1, further comprising the step of modifying the locking requirements of the one or more code paths of the software program that access one or more of the data items so as to acquire all N partition locks and the global lock where a global lock would have been acquired prior to accessing of the one or more data items and so as to release all N partition locks and the global lock where a global lock would have been released after accessing of the one or more data items.

3. (original) The method according to claim 1, wherein the identified code path includes a plurality of branches, and wherein said optimizing includes optimizing the locking requirements of the identified code path so the locks being acquired and released in the code path are those associated with the data items being touched by each branch of identified code path.

4. (original) The method according to claim 3, wherein said optimizing includes optimizing the locking requirements of each branch of the identified code path so the locks being acquired and released in each branch are those associated with the data items being touched by said each branch.

5. (currently amended) The method according to claim 1, further comprising the step of evaluating the software program after said optimizing the locking requirements so as to determine if ~~the~~ overall performance of the software program is acceptable, wherein the overall system performance is based on reducing contention of the highly contended software lock.

6. (original) The method according to claim 5, wherein in the case where said evaluating determines that the overall performance of the software program is not acceptable, then said method includes identifying another code path of the one or more code paths and repeating said steps of determining, partitioning, optimizing, and evaluating for the another identified code path.

7. (original) The method according to claim 1, further comprising the step of first determining a methodology for partitioning the data set.

8. (original) The method according to claim 1, wherein the code path first identified is the heaviest used code path.

9. (original) The method according to claim 6, wherein the code path first identified is the heaviest used code path and wherein the another code path and subsequent code paths are identified sequentially in the direction from the heaviest used code path to a lesser used path.

10. (original) The method according to claim 1, wherein there is one of a plurality or a multiplicity of code paths that access one or more of the data items.

11. (currently amended) A method for reducing contention of a highly contended software lock protecting data items of a data set, all of the data items being stored in a system memory of a multi-processor computer system, said method comprising the steps of:

first determining a methodology for partitioning the data set into partitions;

creating N partition locks, one for each partition, where $N \geq 2$;

modifying ~~the~~ locking requirements of each of one or more code paths of a software program that access one or more of the data items so as to acquire all N partition locks and a global lock where the global lock would have been acquired prior to accessing of the one or more data items and so as to release all N partition locks and the global lock where the global lock would have been released after accessing of the one or more data items;

identifying one code path from the one or more code paths of the software program that access one or more of the data items;

next determining which data items of the data set are ~~touched~~ accessible by the identified code path;

partitioning at least one ~~some~~ of the data items that are ~~touched~~ accessible by the identified code path; and

optimizing the locking requirements of the identified code path so the locks being acquired and released in the identified code path are those associated with the data items being ~~touched~~ accessible by the identified code path.

12. (original) The method according to claim 11, wherein the identified code path includes a plurality of branches, and wherein said optimizing includes optimizing the locking requirements of the identified code path so the locks being acquired and released in the code path are those associated with the data items being touched by each branch of identified code path.

13. (original) The method according to claim 12, wherein said optimizing includes optimizing the locking requirements of each branch of the identified code path so the locks being acquired and released in each branch are those associated with the data items being touched by said each branch.

14. (currently amended) The method according to claim 11, further comprising the step of evaluating the software program after said optimizing the locking requirements so as to determine if ~~the~~ overall performance of the software program is acceptable, wherein the overall system performance is based on reducing contention of the highly contended software lock.

15. (original) The method according to claim 14, wherein in the case where said evaluating determines that the overall performance of the software program is not acceptable, then said method includes identifying another code path of the one or more code paths and repeating said steps of determining, partitioning, optimizing and evaluating for the another identified code path.

16. (original) The method according to claim 15, wherein the code path first identified is the heaviest used code path and wherein the another code path and subsequent code paths are identified sequentially in the direction from the heaviest used code path to a lesser used path.

17. (currently amended) A method for reducing contention of a highly contended software lock protecting data items of a data set, all of the data items being stored in a system memory of a multi-processor computer system, said method comprising the steps of:

defining partitions within the data set;

creating N partition locks, one for each partition, where $N \geq 2$;

partitioning ~~some~~ at least one of the data items into the partitions; and

modifying ~~the~~ locking requirements of all code paths of the one or more code paths of a software program that access one or more of the data items so that the locks being acquired and released in each of said all code paths are those associated with the accessible ~~touching~~ data items of each respective partition.

18. (original) The method according to claim 17, wherein one code path of said all code paths includes a plurality of branches, and wherein said modifying includes modifying the locking requirements of each branch of said one code path so the locks being acquired and released in each branch are those associated with the data touching said each branch.

19. (original) The method according to claim 17, further comprising the step of evaluating the software program after said modifying the locking requirements so as to determine if the overall performance of the software program is acceptable.

20. (original) The method according to claim 19, wherein in the case where said evaluating determines that the overall performance of the software program is not acceptable, then said method includes partitioning more data items and repeating said steps of modifying and evaluating.

21. (original) The method according to claim 17, further comprising the step of first determining a methodology for partitioning the data set.

22. (original) The method according to claim 17, wherein there is one of a plurality or a multiplicity of code paths that access one or more of the data items.

23. (currently amended) A method for reducing contention of a highly contended software lock protecting data items of a data set, all of the data items being stored in a system memory of a multi-processor computer system, said method comprising the steps of:

first determining a methodology for partitioning the data set;

defining partitions within the data set based upon the methodology;

creating N partition locks, one for each partition, where $N \geq 2$;

partitioning ~~some~~ a plurality of the data items into the partitions;

modifying ~~the~~ locking requirements of all code paths of the one or more code paths of a software program that access one or more of the data items so that the locks being acquired and released in each of said all code paths are those associated with accessible ~~the touching~~ data items; and

evaluating the software program after said modifying the locking requirements so as to determine if ~~the~~ overall performance of the software program is acceptable, wherein the overall system performance is based on reducing contention of the highly contended software lock.

24. (original) The method according to claim 23, wherein in the case where said evaluating determines that the overall performance of the software program is not acceptable, then said method includes partitioning more data items and repeating said steps of modifying and evaluating.

25. (currently amended) A method for accessing one or more data items in a data set by a software program, all of the data items being stored in a system memory of a multi-processor computer system, said method comprising the steps of:

defining partitions within the data set;

creating N partition locks, one for each partition, where $N \geq 2$;

identifying one code path from one or more code paths of the software program that access one or more of the data items;

determining which data items of the data set are touched by the identified code path;

partitioning at least one ~~some~~ of the data items that are accessible ~~touched~~ by the identified code path;

optimizing ~~the~~ locking requirements of the identified code path so the locks being acquired and released in the identified code path are those associated with the data items being accessible ~~touched~~ by the identified code path;

locking the data items of the data set that are touched by the identified code path while keeping unlocked the data items of the data set that are not being accessible ~~touched~~ by the identified code path;

accessing one or more of the locked data items; and

releasing the locks associated with the locked data.

26. (original) The method according to claim 25, further comprising the step of modifying the locking requirements of the one or more code paths of the software program that access one or more of the data items so as to acquire all N partition locks and the global lock where a global lock would have been acquired prior to accessing of the one or more data items and so as to release all N partition locks and the global lock where a global lock would have been released after accessing of the one or more data items.

27. (original) The method according to claim 25, wherein in the case where accessing data in the one or more code paths in which all N partition locks and the global lock are acquired and released, said method further comprised the steps of:

acquiring all N partition locks and the global lock;

accessing the data being protected by the acquired N partition locks and the global lock; and

releasing all N partition locks and the global lock.

28. (original) The method according to claim 25, wherein the identified code path is the heaviest used code path that accesses data items.

29. (original) The method according to claim 25, wherein locking requirements for a plurality of code paths are optimized, and wherein said acquiring, accessing and releasing are selectively effected in any one of the plurality of code paths provided that the data items to be locked in said any one code path are not locked in any other of the plurality of code paths.